(19) RÉPUBLIQUE FRANÇAISE

INSTITUT NATIONAL
DE LA PROPRIÉTÉ INDUSTRIELLE

PARIS

(11) N° de publication : **2 767 939**
(à n'utiliser que pour les commandes de reproduction)

(21) N° d'enregistrement national : **98 08058**

(51) Int Cl[6] : **G 06 F 12/08**, G 06 F 15/16

(12) **BREVET D'INVENTION** **B1**

(54) PROCEDE D'ALLOCATION DE MEMOIRE DANS UN SYSTEME DE TRAITEMENT DE L'INFORMATION MULTIPROCESSEUR.

(22) **Date de dépôt :** 25.06.98.

(30) **Priorité :** 04.09.97 FR 09711025.

(43) **Date de mise à la disposition du public de la demande :** 05.03.99 Bulletin 99/09.

(45) **Date de la mise à disposition du public du brevet d'invention :** 02.11.01 Bulletin 01/44.

(56) **Liste des documents cités dans le rapport de recherche :**

Se reporter à la fin du présent fascicule

(60) **Références à d'autres documents nationaux apparentés :**

(71) **Demandeur(s) :** *BULL SA Société anonyme* — FR.

(72) **Inventeur(s) :** BORDAZ THIERRY, ROMAND PATRICE et SORACE JEAN DOMINIQUE.

(73) **Titulaire(s) :**

(74) **Mandataire(s) :**

FR 2 767 939 - B1

2767939

PROCEDE D'ALLOCATION DE MEMOIRE DANS UN SYSTEME DE TRAITEMENT DE L'INFORMATION MULTIPROCESSEUR

La présente invention concerne un procédé d'allocation de mémoire dans un système de traitement de l'information multiprocesseur, plus particulièrement un procédé d'allocation d'une mémoire à accès non uniforme.

Dans le cadre de l'invention, le terme "non uniforme" s'entend dans un sens temporel, comme il va l'être montré. De même, le terme "une mémoire" s'entend dans un sens général. Il peut signifier une mémoire distribuée, une hiérarchie de mémoire (par exemple comprenant des bancs de mémoires à temps d'accès différents), ou un ensemble de mémoires de types différents.

Comme il est bien connu dans le domaine informatique, il est possible d'augmenter la puissance d'une machine en augmentant le nombre de processeurs dont elle est composée. Un type de machine connu sous le nom "SMP" (de l'anglo-saxon "Symetrical MultiProcessor" ou multiprocesseur symétrique) permet aux différents processeurs d'une même machine d'accéder de façon symétrique à sa mémoire au moyen d'un bus système. Ce sont des machines avec mémoire à accès uniforme dans la mesure où le temps d'accès à la mémoire est sensiblement le même pour toutes les données accédées.

Pour cette raison, l'architecture est dite "UMA" (de l'anglo-saxon "Uniform Memory Access" ou accès uniforme à la mémoire).

La figure 1 annexée à la présente description illustre schématiquement un exemple d'architecture du type "UMA".

Le système de traitement de l'information 1, que l'on appellera ci-après module "SMP", comprend un certain nombre d'unités centrales ou processeurs, ou encore "CPU"

selon la terminologie anglo-saxonne. On a représenté quatre unités centrales dans l'exemple de la figure 1 : 10 à 13. On associe à ces unités centrales 10 à 13, une mémoire centrale 14 accessible par tous.

Puisque tous les accès s'effectuent à l'intérieur du module 1, c'est-à-dire en local, et si l'espace mémoire total disponible présente une homogénéité quant au temps d'accès (ce qui constitue l'hypothèse de départ, puisqu'il s'agit d'une architecture "UMA"), le temps d'accès reste sensiblement le même, quelle que soit l'unité centrale 10 à 13 qui effectue une requête.

Bien que, sur la figure 1, il ait été représenté quatre unités centrales 10 à 13, il doit être clair que ce nombre est tout à fait arbitraire. Il peut être augmenté ou diminué. Cependant, la courbe de performances de telles machines ne croît pas de façon linéaire en fonction du nombre de processeurs. Un nombre élevé de processeurs fait que le système consomme plus de temps pour des problèmes d'accessibilité à ses ressources qu'il n'en dispose pour exécuter des applications. Ceci a pour conséquence d'infléchir considérablement la courbe de performances lorsque le nombre de processeurs dépasse une valeur optimale, souvent estimée à quatre environ. L'état de la technique propose différentes solutions à ce problème.

Une solution connue consiste à regrouper en grappes plusieurs machines de façon à les faire communiquer entre elles au moyen d'un réseau. Chaque machine possède un nombre optimal de processeurs, par exemple quatre, et son propre système d'exploitation. Elle établit une communication avec une autre machine toutes les fois qu'elle effectue un traitement sur des données détenues à jour par cette autre machine. Le temps nécessaire à ces communications et la nécessité de travailler sur des données cohérentes posent des problèmes de latence pour des applications volumineuses telles que, par exemple, les applications réparties qui

demandent de nombreuses communications. La latence est la durée qui sépare l'instant d'émission d'une requête d'accès à la mémoire et l'instant auquel la réponse à cette requête est reçue.

Une autre solution connue est celle des machines à architecture de type dit "NUMA" (de l'anglo-saxon "Non Uniform Memory Access"). Ce sont des machines avec mémoire à accès non uniforme, dans la mesure où le temps d'accès à la mémoire varie selon la localisation des données accédées. Une machine de type "NUMA" est constituée de plusieurs modules, chaque module comprenant un nombre optimal de processeurs et une partie physique de la mémoire totale de la machine. Une telle machine est à accès mémoire non uniforme car un module accède généralement plus facilement et plus rapidement à une partie physique de la mémoire qu'il ne partage pas avec un autre module qu'à une partie qu'il partage. Bien que chaque module possède un bus système privé reliant ses processeurs et sa mémoire physique, un système d'exploitation commun à tous les modules permet de considérer l'ensemble des bus systèmes privés comme un seul et unique bus système de la machine. Un adressage logique affecte un lieu de résidence à un emplacement de mémoire physique déterminé d'un module. Pour un processeur considéré, on distingue les accès à une partie de mémoire locale, située physiquement sur le même module que le processeur, et les accès à une partie de mémoire distante, située physiquement sur un ou plusieurs autres modules que celui où est situé le processeur.

La figure 2 annexée à la présente description illustre schématiquement un exemple d'architecture de ce type, c'est-à-dire une architecture "NUMA". Pour simplifier le dessin, on a supposé que le système de traitement de l'information 1' comprend seulement deux modules, M_a et M_b, du type "SMP" précité, et que les deux modules sont identiques. Il doit cependant être bien compris que le système de traitement de l'information 1' peut comporter un

plus grand nombre de modules et que les modules M_a et M_b, peuvent être différents (notamment en ce qui concerne le nombre d'unités centrales).

Le module M_a comprend donc quatre unités centrales 10_a à 13_a, et une mémoire centrale 14_a. De même, le module M_b comprend quatre unités centrales 10_b à 13_b, et une mémoire centrale 14_b. Les deux mémoires 14_a et 14_b, (et de façon plus générale les n mémoires centrales) communiquent entre elles à l'aide de ce qui est appelé un "lien" 2, généralement via des antémémoires dites éloignées 15_a et 15_b, respectivement. Le lien 2 ne se résume pas à de simples liaisons physiques, mais comprend des circuits électroniques divers classiques (circuits de commande, d'interface, etc.), qu'il est inutile de décrire plus avant.

On comprend aisément que, dans une telle architecture, si une application s'exécute dans le module M_a, par exemple, le temps d'accès à la mémoire "proche" 14_a (accès en local) est, *a priori*, inférieur au temps d'accès à la mémoire "éloignée" 14_b située dans le module M_b, ce quelle que soit l'unité centrale 10_a à 13_a, concernée. Il est notamment nécessaire de passer par le lien 2 lorsque les données sont physiquement stockées dans un autre module, ce qui augmente sensiblement le temps de transfert.

Dans les systèmes de traitement de l'information modernes, l'allocation de la mémoire pour une application donnée s'effectue sur la base d'un espace mémoire virtuel. Cette allocation est placée sous la commande du système d'exploitation ou "OS" (de l'anglo-saxon "Operating System"). On effectue ensuite une correspondance dynamique entre l'espace mémoire virtuel et la mémoire physique. Pour ce faire, on utilise classiquement des tables de correspondance d'adresses. On parle de "mapping" dynamique, selon l'expression anglo-saxonne couramment utilisée. Différents types de configurations de mémoires ont été

proposés : organisation par régions ou par segments. Pour fixer les idées, dans ce qui suit, sans limiter en quoi que ce soit la portée de l'invention, on se placera dans le cas d'une configuration du type "segment". De façon pratique, un segment se définit comme un espace d'adresses virtuelles contiguës, de longueur fixe et déterminée.

De façon plus précise, dans l'art connu, la correspondance dynamique précitée ou "mapping" s'effectue selon des règles communes à toutes les applications quels que soient leurs types, sans tenir compte de la localisation de la mémoire physique. De façon pratique, si un processus veut accéder à une adresse virtuelle et qu'aucune entrée dans la table de correspondance d'adresses n'est trouvée, il y a génération d'une exception qui se formalise par la détection d'un défaut de page, selon la terminologie "UNIX" (marque déposée). Le terme "page" peut être défini de façon plus générale comme étant une "plage d'adresses contiguës". Une page constitue une subdivision d'un segment. Cependant, pour des raisons de simplification, le terme "page" sera utilisé dans ce qui suit. Suite à une détection de défaut de page, un dispositif appelé gestionnaire attribue de la mémoire physique, et ce selon les règles communes précitées. Cette méthode d'allocation simple est tout-à-fait adaptée pour les machines classiques "SMP" du type "UMA" précité, puisque le temps moyen d'accès à la mémoire est uniforme.

Par contre, lorsqu'il s'agit d'une architecture du type "NUMA", telle que décrite en regard de la figure 2, pour laquelle le temps d'accès n'est plus uniforme, le besoin se fait sentir de disposer d'un procédé d'allocation de mémoire qui minimise l'impact négatif sur les performances du système.

Dans l'art connu, des procédés ont été proposés en ce sens. A titre d'exemple, on a proposé de modifier les règles d'allocation de mémoire en vue d'obtenir une optimisation, mais les règles une fois modifiées restent

identiques pour toutes les applications. En outre, ce procédé présente des inconvénients supplémentaires. Les règles modifiées peuvent s'avérer avantageuses pour une application donnée, mais inappropriées, voire dangereuses pour une autre.

On a également proposé des "API" particuliers (de l'anglo-saxon "Application programmable Interface" ou interface programmable pour application) adaptées pour définir un algorithme particulier associé à une application donnée, en vue d'effectuer la correspondance ("mapping") entre l'espace mémoire virtuel et la mémoire physique, mais il est alors nécessaire de modifier, à la fois, les applications correspondantes et la partie résidente du système d'exploitation ("kernel"). Cette méthode ne peut donc pas s'appliquer telle quelle aux programmes existants. En tout état de cause, elle manque de souplesse et son efficacité est limitée.

L'invention se fixe donc pour objet un procédé d'allocation de mémoire pour un système de traitement de l'information à accès non uniforme de la mémoire centrale, notamment du type "NUMA" précité, qui vise à répondre aux besoins qui se font sentir pour cette architecture particulière et qui ne présente pas les inconvénients des procédés de l'art connu.

Pour ce faire, l'allocation de mémoire s'effectue en fonction du profil propre à chaque application, c'est-à-dire en mettant en oeuvre un jeu de règles d'allocation tenant compte de ce profil entre lesquelles, à chaque faute de page détectée, il est opéré automatiquement une recherche déterminant laquelle doit être exécutée pour l'allocation de mémoire physique.

Dans un mode de réalisation préféré, l'allocation de mémoire mémoire physique s'effectue en outre, en tenant compte du type de faute de page. En effet, lors de son exécution, une application se subdivise en différents objets

tels que du texte, des données, de la mémoire partagée, etc., qui utilisent différemment l'espace global de mémoire du système. L'invention permet donc d'optimiser également les accès mémoire en fonction de ce paramètre.

Le procédé de l'invention, dans les variantes qui viennent d'être mentionnées, offre une amélioration très significative par rapport à l'art connu, et notamment de meilleures performances et une grande souplesse. En outre, il n'est pas nécessaire de modifier les applications existantes. Cependant, il doit être noté que, dans la pratique, l'espace d'adressage virtuel d'un système de traitement de l'information multiprocesseur, notamment de type "NUMA", est habituellement très vaste. Pour fixer les idées, si le système d'exploitation est sous l'environnement "UNIX" précité ou une de ses variantes, un simple segment de mémoire virtuelle représente habituellement 256 MO. On conçoit aisément, dans ces conditions, qu'une règle unique par segment peut ne pas être optimisée pour un certain nombre d'applications, même si ce segment est associé à une seule classe : "type données", par exemple. Il est par ailleurs usuel de subdiviser un segment en sous-espaces d'adressage virtuel, que l'on appellera ci-après "plages virtuelles" ou encore "ranges" selon la terminologie anglo-saxonne. Les différentes zones d'adresses d'un segment, correspondant à ces plages virtuelles, peuvent être utilisées de façon différente.

Contrairement à une page, qui elle aussi forme une subdivision d'un segment, les "plages virtuelles" ainsi définies peuvent avoir des longueurs variables. De façon pratique, on admettra que, dans l'exemple d'application préférée (environnement "UNIX"), la granularité des plages virtuelles peut descendre jusqu'au niveau de la page.

Pour fixer les idées, on peut, par exemple, réserver une plage virtuelle de 50 MO pour un tableau définissant une mémoire physique tampon en vu de lire des données

enregistrées sur un disque, par l'intermédiaire d'un contrôleur d'entrée-sortie.

Même si on se limite à cet exemple simple, on comprend que la localisation de la mémoire tampon dans l'une des mémoires physiques du système, dans le cas d'une architecture "NUMA", par rapport d'une part au module auquel est rattaché le disque précité, et d'autre part à l'application spécifique utilisant ces données, n'est pas indifférente, en terme de performances.

Aussi, selon une variante supplémentaire de l'invention, et toujours selon un mode de réalisation préféré, on associe sélectivement une politique d'allocation de mémoire spécifique à chacun des plages virtuelles ou "ranges". Cette caractéristique technique permet d'optimiser encore plus les allocations de mémoire en cas de détection de faute de page, au prix de modifications limitées apportées aux applications pour lesquelles cette variante du procédé de l'invention est mise en oeuvre.

Selon cette variante, lorsqu'il y a génération d'une exception, qui se traduit par une faute de page, le gestionnaire chargé de trouver la règle à exécuter scrute une table en vue de déterminer s'il existe une politique d'allocation spécifique associée à cette plage virtuelle. Si cette politique spécifique n'existe pas, la politique gouvernant l'ensemble hiérarchique supérieur que constitue le segment est adoptée. Dans le cas contraire, une règle d'allocation de mémoire est dérivée de cette politique spécifique. En d'autres termes, un même segment peut comprendre une ou plusieurs plages virtuelles, ou "ranges", associées chacune à une politique d'allocation de mémoire spécifique, en même temps qu'une ou plusieurs autres plages virtuelles obéissant à la politique générale du segment, voire du système, et aux règles qui en découlent. Il doit être clair que le terme "spécifique" n'implique pas obligatoirement que la politique d'allocation associée à une

plage donnée, repérée arbitrairement *n* (avec *n* compris entre 1 et *m*, si le segment comprend *m* plages), soit différente de celle associée à une ou plusieurs autres plages virtuelles, par exemple les plages virtuelles *n*+2 et *n*+5.

De façon pratique, on recourt à des structures de liste pour déterminer quelle règle d'allocation de mémoire il est nécessaire d'utiliser, chaque élément de la liste correspondant à une plage d'adresses contiguës, délimitée par les adresses de début et de fin de la plage virtuelle. De façon pratique, un élément de la liste est un emplacement de mémoire stockant les règles d'allocation de mémoire qui s'appliquent à une plage virtuelle donnée. Les différents éléments de la liste sont scrutés, séquentiellement, pour déterminer quelles règles doivent être appliquées.

Ce processus peut encore être accéléré. Selon une variante supplémentaire de ce mode de mise en oeuvre de l'invention, on subdivise les segments en *N* sous-espaces d'adresses virtuelles contigus, tous de même longueur. On prévoit une table, dite de "hash", comprenant également *N* entrées. À ces *N* entrées sont associées autant de structures individuelles de listes. La longueur des structures de liste individuelles n'est pas fixe. Elle dépend du nombre de plages virtuelles associées à cette entrée. Lorsque une faute de page est détectée, le gestionnaire d'adressage *H* connaît l'adresse qui a provoquée la faute de page. Il lui est donc aisé de déterminer le numéro de l'entrée correspondante dans la table.

La table "hash" est constituée de *N* emplacements de mémoire, chaque emplacement stockant au moins une information sur le fait qu'il existe une plage virtuelle associée à cette entrée et qui nécessite le recours à une politique spécifique d'allocation de mémoire. De façon pratique, lorsque une faute de page est détectée, son entrée dans la table est lue et dès lors qu'il existe un pointeur caractéristique de la présence d'une politique d'allocation

spécifique, il s'ensuit, sans étape supplémentaire, la détermination de celle des plages virtuelles précitées du segment qui est associée à cette faute de page. Pour les segments ne comprenant pas de plages virtuelles associées à une politique spécifique, la politique à appliquer est celle du segment dans sa globalité. Par contre, pour les plages virtuelles nécessitant des politiques d'allocation de mémoire spécifiques, il est nécessaire de scruter un ou plusieurs éléments d'une structure de liste associés à une entrée particulière de la table. Cependant, la longueur de cette structure de liste est beaucoup plus limitée que dans le cas précédent, puisqu'elle ne couvre que les plages virtuelles associées à ladite entrée. Du fait de ces deux particularités, la variante de réalisation qui vient d'être décrite accélère bien le processus d'allocation de mémoire, pour le moins la phase de détermination des règles à appliquer.

L'invention a donc pour objet un procédé d'allocation d'emplacements de mémoire physique par mise en correspondance avec au moins une plage d'adresses contiguës de mémoire dans un espace d'adressage virtuel associée à une application logicielle déterminée, l'application étant en cours d'exécution dans un système de traitement de l'information comprenant une unité de mémoire à accès non uniforme et utilisant plusieurs types de mémoire virtuelle, ladite mise en correspondance s'effectuant par scrutation d'une table de correspondance d'adresses, caractérisé en ce qu'il comprend au moins une étape consistant à lier ladite application logicielle déterminée à des règles d'allocation de mémoire choisies parmi un jeu de règles prédéfinies, et en ce que, lorsque ladite table de correspondance d'adresses ne comporte pas d'entrée pour une plage d'adresses contiguës de mémoire d'adresses virtuelles associées à ladite application logicielle déterminée, il comprend une étape de génération d'une exception et une étape subséquente d'allocation d'un emplacement de mémoire physique selon une desdites règles d'allocation de mémoire, le choix de la

règle étant assujetti au profil desdits types de mémoire virtuelle utilisés par l'application logicielle déterminée.

L'invention sera mieux comprise et d'autres caractéristiques et avantages apparaîtront à la lecture de la description qui suit en référence aux figures annexées, parmi lesquelles :

- la figure 1 illustre schématiquement une architecture de système de traitement de l'information à accès de mémoire uniforme dite "UMA" ;

- la figure 2 illustre schématiquement une architecture de système de traitement de l'information à accès de mémoire non uniforme dite "NUMA" ;

- les figures 3a et 3b illustrent schématiquement les accès à la mémoire pour deux exemples d'applications logicielles ;

- la figure 4 illustre un exemple de subdivision d'une application logicielle ;

- la figure 5 illustre schématiquement l'allocation d'un emplacement de mémoire selon l'art connu, lors de la génération d'une faute de page ;

- les figures 6a et 6b illustrent schématiquement l'allocation d'un emplacement de mémoire selon l'invention, lors de la génération d'une faute de page ;

- les figures 7a et 7b illustrent un mode de réalisation supplémentaire du procédé selon l'invention ;

- la figure 8 illustre schématiquement un exemple pratique d'implantation des variantes du procédé selon l'invention dans un système de traitement de données numérique ;

- la figure 9 illustre schématiquement un exemple pratique d'implantation du mode de réalisation supplémentaire du procédé selon l'invention, selon une première variante ;

- et les figures 10a à 10c illustrent schématiquement un exemple pratique d'implantation du mode de réalisation supplémentaire du procédé selon l'invention, selon une seconde variante.

Pour fixer les idées, sans limiter en quoi que ce soit la portée de l'invention, on se placera ci-après dans le contexte d'un système de traitement de l'information dont le système d'exploitation est du type "UNIX" ou similaire, sauf mention contraire.

Pour les applications fonctionnant sous cet environnement, l'espace d'adressage virtuel peut être divisé en différents types, notamment les types suivants :

- le texte ou le texte programme (code exécutable) ;
- les données initialisées ;
- les données modifiées ;
- les "stacks" ou piles ;
- le "heap", c'est-à-dire l'allocation dynamique (tables, etc.) ;
- la mémoire partagée ;
- les bibliothèques partagées.

Lors du déroulement d'une application, celle-ci utilise les différents types de mémoire du système également de façon différente. En outre, une application ou une nouvelle instance de la même application peut s'exécuter dans l'un ou l'autre des deux modules M_a ou M_b du système de la figure 2. Ce qui est vrai pour une même application, l'est encore plus pour deux applications de profils différents.

Les figures 3a et 3b illustrent schématiquement deux types d'applications, à savoir une "mini-base de données" ("minidatabase") et une base de données plus traditionnelle.

On a représenté sur ces deux figures la mémoire globale du système par la référence unique *Mem*.

Dans le premier cas, illustré par la figure 3a, lors d'une période d'initialisation, les accès sont cantonnés à un espace d'adressage représenté symboliquement par la zone *Zini*, située arbitrairement sur la gauche de la figure 3a. Puis, les accès s'effectuent dans un espace d'adressage, symbolisé par une zone *Zf*, également d'étendue restreinte et supposée connexe à la zone *Zini*, dans l'exemple de la figure 3a. Les emplacements de mémoire physique, pour cette application particulière, peuvent donc être cantonnés dans un seul module, et plus précisément en local.

Ce n'est généralement pas le cas pour une base de données classique, comme illustré par la figure 3b. Les accès peuvent s'étendre à tout l'espace mémoire, comme le symbolisent les flèches représentées sur la figure 3b. Il s'ensuit que les emplacements de la mémoire physique occupée sont généralement distribués sur deux modules ou plus.

En outre, comme il a été indiqué, pour une même application, l'espace mémoire virtuel se subdivise en différents types de segments : texte, données, etc.

A titre d'exemple non limitatif, lorsqu'une application donnée *Appli* s'exécute, ses différents composants sont partitionnés en segments de mémoire virtuels : Texte *T*, Données *Da* (de différents types), Stack *St*, mémoire partagée *Shm*, fichiers *Fi*, etc., comme illustré par la figure 4. Classiquement, un dispositif gestionnaire *H*, ou "handler" selon la terminologie anglo-saxonne couramment utilisée, attribue à ces segments de mémoire virtuels des emplacements dans la mémoire physique globale *Mem* du système.

On va maintenant décrire le mécanisme d'allocation de mémoire physique sur détection d'une faute de page.

On a indiqué précédemment qu'une application en cours d'exécution utilise les différents type de mémoire de façon différente également. De même, une application qui se déroule initialement dans un module donné (par exemple figure 2 : M_a) peut se continuer dans un autre (par exemple figure 2 : M_b), ou une instance supplémentaire de cette application peut se créer et s'exécuter dans un module différent.

Si on suppose qu'une application tente d'effectuer une instruction particulière, par exemple une instruction de chargement, ou "load", à une adresse virtuelle déterminée, par exemple l'adresse arbitraire "0_X 2000", l'unité centrale (par exemple figure 2 : 10a), dans laquelle se déroule le processus en cours, décode l'instruction et une table de correspondance d'adresses (non représentée) va être scrutée. Si l'entrée recherchée ne s'y trouve pas, il y a émission d'une exception qui se traduit par une faute de page détectée par le gestionnaire *H*. Il est donc nécessaire, dans ces circonstances, d'attribuer un emplacement de mémoire physique pour l'adresse virtuelle "0_X 2000" ci-dessus.

Dans l'art connu, il n'existe qu'un seul type d'allocation. En d'autres termes, les règles utilisées sont uniques quel que soit le profil de l'application et le type de segment. Le gestionnaire *H* affecte donc un emplacement de mémoire physique conformément aux règles d'allocation utilisées par le système. Par exemple, conformément à ces règles, l'allocation s'effectue systématiquement dans la mémoire physique locale, c'est-à-dire dans la mémoire 14_a si le processus se déroulait dans le module M_a sous la conduite d'une des unités centrales 10_a à 13_a. Cette règle peut s'avérer intéressante pour un segment de type texte, mais non optimisée pour d'autres types de segments.

Le mécanisme ci-dessus est illustré par la figure 5. L'application *Appli* génère une faute de page *Fp* et le gestionnaire *H* attribue un emplacement de la mémoire

physique *Mem* (dont les partitions, Z_1 à Z_n, ont été symbolisées par des traits en pointillé sur la figure 5), selon des règles prédéfinies.

Comme il a été indiqué également, ces règles peuvent être modifiées, mais elles restent les mêmes pour toutes les applications et tous les types de segments.

Tout au contraire, selon le procédé de l'invention, l'allocation de la mémoire physique *Mem* va être réalisée conformément à un jeu de règles qui tient compte, d'une part, du profil de l'application, et d'autre part, dans un mode de réalisation préféré, du type de faute de page.

Les figures 6a et 6b illustrent le mécanisme d'allocation de la mémoire physique selon l'invention.

Selon une caractéristique principale du procédé selon l'invention, on lie chaque application à un jeu de règles d'allocation prédéfinies. Pour ce faire, il est nécessaire d'associer chaque application $Appli_X$ (x étant un indice arbitraire) à un profil particulier. Cette association s'effectue sous la conduite du système d'exploitation, ou "OS", qui le mémorise. La figure 6a illustre schématiquement le mécanisme de l'association.

On a représenté, sur cette figure 6a, une application particulière $Appli_X$. Comme il a été indiqué, cette application $Appli_X$ utilise divers types de mémoire : texte, données, etc. Sur la figure 6a, on a représenté six types de mémoire, que l'on a référencés tyM_1 à tyM_6. On lie chaque type de mémoire, tyM_1 à tyM_6, à une règle d'allocation, parmi un jeu de règles prédéfinies que l'on précisera ci-après. Ces règles ont été référencées R_1 à R_6, étant entendu qu'il ne s'agit pas forcément d'un jeu de règles disjointes. En d'autres termes, à titre d'exemple, les règles R_2 et R_3 pourraient être identiques, même si les types de mémoire tyM_2 et tyM_3 sont eux distincts.

Le profil d'allocation de mémoire Pa_X qui vient d'être défini est lié par une association A_X à une application particulière $Appli_X$. Le profil Pa_X est donc une table à deux entrées : types de mémoire tyM_i et règles R_j choisies parmi un jeu de règles prédéfinies, i et j étant des indices arbitraires.

De façon générale, on peut définir une fonction association comme suit :

Association_pa($Appli_X$, Pa_X).

Le profil d'allocation de mémoire lié à une application donnée peut être défini à l'initialisation de l'exécution de cette application ou, de façon dynamique, redéfini à tout moment pendant l'exécution, ce qui augmente la souplesse du procédé.

Sur la figure 6b, les règles d'allocation prédéfinies ont été repérées sous la référence générale *Rg*. Lors de l'apparition d'une exception qui se traduit par une faute de page, *Fp*, c'est-à-dire lorsque la table de correspondance d'adresses ne contient pas d'entrée pour une adresse virtuelle, le gestionnaire *H* recherche le profil Pa_X de l'application $Appli_X$ tel qu'il vient d'être défini. Il détermine aussi, dans un mode de réalisation préféré, le type de faute de page *Fp*. A partir de ces deux paramètres, il attribue des emplacements en mémoire physique, Z_1 à Z_n, soit locaux (dans le même module), soit distants (dans un autre module), soit encore répartis sur l'ensemble de la mémoire *Mem*. Cette répartition, dépendant du profil Pa_X de l'application $Appli_X$ et du type de faute de page *Fp*, est symbolisée, sur la figure 6b, par des flèches multiples (contrairement à la flèche unique du procédé selon l'art connu représenté sur la figure 5).

La fonction d'adressage F_{ad} peut donc se formaliser de la façon suivante :

$F_{ad} = F(Pa_X, \text{Type } Fp)$.

Le choix de la règle à appliquer pour l'allocation de mémoire est donc le résultat de la combinaison de deux paramètres.

De façon plus précise, une application donnée *Applix* spécifie quelles règles d'allocation elle requière pour chaque type de segment de mémoire virtuelle, parmi un jeu de règles prédéfinies. A titre d'exemple, les types de segments suivants sont couramment utilisés : segments "clients", segments de "mapping", segments "permanents", segments de "mémoire de travail" et segments de "bibliothèques partagées".

Pour fixer les idées et sans que cela soit limitatif en quoi que ce soit de la portée de l'invention, on peut définir le jeu de règles suivant, que l'on repère par les codes précisés ci-dessous :

- "*P_STRIPE*" : allocation en bandes, parmi tout l'espace mémoire physique formant la ressource d'une application donnée, réparties dans la mémoire locale (même module) ou distante (modules différents), selon une méthode de type dit "round robin" ;

- "*P_LOCAL*" : la mémoire physique allouée est dans le même module que l'unité centrale ayant provoqué la faute de page ;

- "*P_FIXE*" : la mémoire physique allouée est dans un module prédéfini ;

- "*P_DEFAULT*" (ou "*P_NONE*"): il n'y a pas de règle d'allocation de mémoire physique, et l'on utilise alors des règles par défaut propres au système.

A titre d'exemple, l'allocation du type "*P_STRIPE*", définie ci-dessus, convient *a priori* pour des segments de type "mémoire partagée", alors que l'allocation de type "*P_LOCAL*" convient *a priori* pour des segments du type "mémoire de travail".

Le procédé selon l'invention permet ainsi d'optimiser au mieux les allocations de mémoires physiques en fonction des besoins réels des applications, plus précisément des profils particuliers des applications. Les performances du système de traitement de l'information s'en trouvent améliorées, car les temps d'accès à la ressource mémoire sont aussi optimisés, pour le moins si l'on raisonne en temps moyens d'accès. Un autre avantage est la possibilité de lier une application quelconque au jeu de règles d'allocation de mémoire prédéfinies sans qu'il soit nécessaire de la modifier ou de la recompiler, comme ce serait le cas si on utilisait une nouvelle "API", ainsi qu'il a été indiqué.

En outre, le procédé présente une grande souplesse. Il permet notamment de pouvoir fonctionner selon l'art connu. Par exemple, si des règles d'allocation ne sont pas spécifiées ou requises par une application donnée, des règles par défaut venant du système peuvent être utilisées ("*P_DEFAULT*"). D'autre part, une application "fille" peut "hériter" des règles d'allocation de mémoire associées à l'application "mère" qui l'a créée. Il peut en être de même d'une instance supplémentaire d'une application, qui se déroule par exemple dans un module différent.

Comme il est aisé de le constater, le procédé de l'invention, dans les variantes qui viennent d'être décrites, offre une amélioration significative par rapport à l'art connu, et notamment de meilleures performances et une grande souplesse.

Cependant, comme il a été indiqué dans le préambule de la présente description, l'espace d'adressage virtuel des systèmes de traitement de l'information multiproceseurs actuels peut être extrêmement vaste. Dans le cadre de l'environnement "UNIX", un simple segment représente par exemple 256 MO, soit 2^{16} pages. Aussi, il est d'usage de subdiviser les segments, en tant que de besoin, en "ranges"

ou plages virtuelles, de longueurs variables. Ces sous-espaces virtuels, même s'ils appartiennent à un segment commun, peuvent être utilisés de manière différente par les diverses applications auxquelles ils sont liés. Il s'ensuit également que, pour certains types d'applications au moins, la mise en oeuvre de règles uniques pour la totalité d'un ou plusieurs segments qu'elles utilisent peut ne pas s'avérer entièrement optimisée.

A titre d'exemple non limitatif, on va considérer de nouveau un système de type "NUMA", par référence à la figure 7a. Ce système, référencé 1', est semblable au système 1 de la figure 2, mais il est supposé comprendre au moins trois modules "SMP", M_a, M_b et M_c, interconnectés par un lien 2'. On a également supposé que le module M_c était connecté à une unité de disques D, par l'intermédiaire d'un contrôleur et de circuits habituels d'entrée-sortie, sous la référence unique I/O_c.

On suppose enfin qu'une application donnée $Appli_X$ s'exécute dans un des processeurs du module M_a, par exemple le processeur 10_a. Cette application manipule notamment un segment de l'espace d'adressage virtuel du système 1', référencé arbitrairement Sg_X. Ce segment Sg_X est illustré schématiquement par la figure 7b. Dans un but de simplification, on a supposé qu'il ne comporte que cinq plages virtuelles, ou "ranges", référencés Ra_1 à Ra_5. Dans l'exemple décrit, la plage virtuelle Ra_2 est attribuée à l'application $Appli_X$ précitée et, de façon plus précise, elle concerne un tableau dont la capacité est de 50 MO par exemple. Ce tableau est relatif à une mémoire tampon, de même capacité, localisé dans une des mémoires physiques du système 1'. Toujours dans l'exemple décrit, on a supposé que, du fait des règles associées au type de segment Sg_X et au profil de l'application $Appli_X$, l'emplacement de mémoire tampon était physiquement localisé dans la mémoire centrale 14_b du module M_b. Il s'ensuit que la lecture de données par l'application $Appli_X$ nécessite, notamment, deux transits sur

le lien 2', transits qui constituent des opérations particulièrement pénalisantes en termes de temps de traitement.

Pour éviter la nécessité d'un double transit, il eut été judicieux que les règles d'allocation retenues pour la plage virtuelle Ra_2 imposent une localisation de la mémoire tampon de 50 Mo, soit dans la mémoire centrale physique du module M_C (près des circuits I/O_C et de l'unité de disques D), soit dans la mémoire centrale physique du module M_a, module où s'exécute l'application $Applix$. L'expérience montre, qu'en général, la première solution donne de meilleurs résultats, d'un point de vue performances.

Sur ce simple exemple, il est aisé de constater que, même si on met en oeuvre des règles d'allocation de mémoire adaptées au profil des applications, conformément au procédé de l'invention, voire modifiables dynamiquement, il subsiste des cas où le procédé n'est pas entièrement optimisé.

Aussi, selon un aspect supplémentaire du procédé selon l'invention, dans un mode de réalisation préféré, on associe sélectivement les plages virtuelles, ou "ranges", à des règles spécifiques.

Si on se reporte de nouveau au diagramme de la figure 7b, on a considéré que seules les plages virtuelles Ra_2 et Ra_4 étaient associées à des règles spécifiques. Pour fixer les idées, la plage virtuelle Ra_4 peut également représenter un tableau, mais cette fois-ci de 100 MO, par exemple. Les autres plages virtuelles (représentées en traits hachurés sur la figure 7b) ne sont pas liées à des règles spécifiques. Dans ce cas, ce sont les règles associées à l'unité de mémoire virtuelle hiérarchiquement supérieure, en l'occurrence le segment Sg_X, qui s'appliquent. Ceci s'effectue de la façon qui a été précédemment explicitée.

De façon plus précise, selon ce mode supplémentaire de réalisation, on associe à chaque plage virtuelle des informations supplémentaires définissant une politique spécifique d'allocation de mémoire physique. Toutefois, cette politique est optionnelle. En effet, les informations supplémentaires comprennent un premier champ, que l'on appellera le pointeur "*prange*", indiquant si, pour un segment, on doit appliquer effectivement au moins une politique spécifique ("*prange*" ≠ 0) pour une plage virtuelle ou "range", ou si c'est la politique d'allocation qui lui est associée globalement qui doit être appliquée ("*prange*" = 0).

Un deuxième champ concerne le type de politique. On retrouve, au niveau de la plage virtuelle, le jeu de règles précédemment énoncé pour un segment global : "P_STRIPE", "P_FIXE", etc.

Enfin, au moins pour certains types de politique d'allocation de mémoire, il est nécessaire de préciser le numéro ou l'adresse du module dans lequel la mémoire physique sera allouée. Pour ce faire, il existe un troisième champ ou champ "N° de module". C'est le cas pour le type "P_FIXE" : il est nécessaire de préciser le numéro du module prédéfini. Pour le type "P_STRIPE", il est en outre nécessaire de connaître le numéro du module précédemment utilisé pour l'incrémenter selon la loi de distribution en bandes de mémoire utilisée ("round robin"). Il est donc nécessaire de mémoriser le numéro précédemment utilisé dans une position mémoire, un registre ou un compteur.

Si on se reporte de nouveau au diagramme de la figure 7b, les politiques d'allocation des plages virtuelles Ra_1 à Ra_5 pourraient se décliner comme suit :

- pour les plages virtuelles Ra_1, Ra_3 et Ra_5, pas de politique spécifique, les règles étant celles du segment Sg_X, par exemple type = P_FIXE et numéro de module = N°de M_a ;

- pour la plage virtuelle Ra_2, existence d'une politique spécifique, avec type = P_FIXE et numéro de module = N° de M_C, comme il a été indiqué ci-dessus ;

- pour la plage virtuelle Ra_4, existence d'une politique spécifique, avec, par exemple, type = P_FIXE aussi et numéro de module = N° de M_b.

Le procédé, dans la variante qui vient d'être décrite, permet d'optimiser au mieux l'allocation de mémoire physique sur détection d'une faute de page. Il nécessite, comme dans les variantes précédentes une modification du système d'exploitation, mais aussi des modifications légères des applications qui y font appel.

A titre d'exemple, si on considère des instructions de lecture et d'écriture, du type "*bind*" en terminologie "UNIX", qui doivent s'exécuter dans le module numéro 3 (soit, par exemple, le module M_C), avec un tampon de 50 MO comme indiqué précédemment, il est nécessaire d'ajouter, dans le flot d'instructions, une instruction initialisant, sur appel système, une politique spécifique pour la plage virtuelle utilisée (par exemple Ra_2), instruction que l'on appellera "*policy*". Celle-ci peut prendre la forme suivante :

policy[*adresse*, *taille* (par exemple 50 MO), *politique* (par exemple P_FIXE), *module* (par exemple N° 3)]

Bien que toutes applications puissent tirer profit de cette variante de réalisation supplémentaire du procédé selon l'invention, il n'est cependant pas nécessaire de modifier tous les types d'applications.

Il est utile de noter que celles qui ne sont pas modifiées peuvent s'exécuter normalement. La politique d'allocation de mémoire physique s'effectue de la manière décrite en regard des figures 6a et 6b. Les règles utilisées dépendent notamment du profil de ces applications et sont

avantageusement celles définies précédemment, encore qu'elles puissent être différentes.

Par contre, le procédé, selon la variante supplémentaire, est particulièrement avantageux pour les applications qui manipulent des données sur un même espace virtuel. A titre d'exemples non exhaustifs, on peut citer les applications du type dit "driver", et notamment les "drivers" de disques et de réseau, c'est-à-dire des programmes de gestion de périphériques ou de réseau. Ces applications font appel à des segments de type "données" dans lesquelles il existe des tampons ou "buffers", plages de mémoire pour lesquelles l'application a la possibilité de définir des politiques spécifiques, au travers d'une instruction que l'on appelera "*system policy*".

D'autres types d'applications sont particulièrement concernés. Il s'agit des applications qui communiquent entre elles par l'intermédiaire de mémoires partagées. Par exemple, si on considère une première application qui s'exécute dans un des processeurs du module M_a de la figure 7a et une deuxième application qui s'exécute dans un des processeurs de ce même module M_a, et partagent une première plage virtuelle d'un segment du type "mémoire partagée", il est intéressant, *a priori* pour des raisons de performances, que le type de politique d'allocation de mémoire soit "P_FIXE" ou "P_LOCAL", et que le numéro de module soit égal à 1 (module M_a). Cette disposition devrait, *a priori*, améliorer les performances du système. Il est donc utile d'affecter une politique spécifique d'allocation de mémoire à cette première plage virtuelle. De même, si une troisième application s'exécute dans le module M_b et partage, avec la première application, une seconde plage virtuelle, appartenant au même segment de type "mémoire partagée", il peut être intéressant que le type de politique d'allocation de mémoire soit "P_FIXE" et que le numéro de module soit égal à 2 (module M_b). Cette disposition devrait aussi, *a priori*, améliorer les performances du système. Il

est donc également utile d'affecter une politique spécifique d'allocation de mémoire à cette seconde plage virtuelle.

On va maintenant décrire, dans un exemple de réalisation pratique, comment le procédé de l'invention, dans ces différentes variantes, peut être implanté sur une machine réelle. Pour fixer les idées, on se placera ci-après, sans que cela limite en quoi que ce soit la portée de l'invention, dans le cadre d'une machine sous environnement "UNIX" ou similaire.

Dans ce type de machine, il existe une table des segments *Scb* dans laquelle sont enregistrées des données définissant ces segments, notamment leurs types ou classes, comme illustré schématiquement sur la figure 8. Le procédé de l'invention tire parti de cette particularité.

Si on considère une application donnée $Applix$, on stocke son profil de mémoire Pa_X dans la structure ayant créé les segments utilisés par cette application. Le profil Pa_X, comme on l'a montré en relation avec la figure 6a, comprend généralement plusieurs types de mémoires. Si on considère un segment donné, son type est décrit par un enregistrement Sgc_Y dans la table des segments *Scb*. Ce type se réfère à un élément du profil Pa_X, qui est décrit à son tour, selon l'invention, par des données supplémentaires MP_Y, enregistrées dans la table *Scb*.

Ces données supplémentaires MP_Y représentent précisément la politique d'allocation de mémoire à appliquer, soit au niveau global du segment Sgc_Y, soit au niveau des plages virtuelles, subdivisions de longueurs variables de ce segment. La figure 8 illustre donc les interactions principales entre une application donnée $Applix$ et la table de segments *Scb*.

En réalité, et dans la mesure où une application a été modifiée pour supporter une politique d'allocation de mémoire spécifique au niveau des plages virtuelles, il

existe plusieurs jeux de données supplémentaires pour chaque segment.

Comme il a été indiqué ci-dessus, les données supplémentaires de chaque jeu comprennent plusieurs champs : le champ du pointeur "*prange*" qui précise s'il existe ou non une politique spécifique à prendre en compte dans le segment, un champ que l'on appellera "*policy_type*" qui précise le type de règle à appliquer ("P_FIXE", etc.) et un champ que l'on appellera "*module*" qui précise le numéro de module, du moins pour certains types de règles (par exemple si la règle est "P_FIXE"). Pour délimiter les différentes plages virtuelles, il est en outre nécessaire de disposer d'informations précisant les adresses virtuelles des bornes basses et hautes de ces plages virtuelles. Ces informations figurent dans des champs que l'on appellera "*pno_start*" et "*pno_end*", respectivement.

Lorsqu'une faute de page est détectée, il est donc nécessaire de balayer ces différentes données pour déterminer la politique d'allocation précise à appliquer. Selon un aspect de l'invention, on adopte une structure de liste pour organiser les jeux de données supplémentaires, comme illustré schématiquement par la figure 9.

On suppose que le segment adressé ayant causé la faute de page comprend *z* plages virtuelles. La première position mémoire de la structure de liste, dans la table *Scb*, comprend des données relatives au segment dans sa globalité, et notamment la politique globale d'allocation de mémoire pour ce segment. Pour une application non modifiée, seule cette position mémoire existe.

Les autres éléments de la liste, référencés Ra_1 à Ra_Z, sont relatifs aux différentes plages virtuelles contiguës. On retrouve donc, pour chacun des éléments, les différents champs : "*policy_type*", "*module*", "*pno_start*" et "*pno_end*" pour les segments dont le pointeur "*prange*" est différent de zéro.

L'adresse dans le segment ayant causé une faute de page est connue. Le gestionnaire *H* fournit cette adresse ainsi que le segment concerné, que l'on appellera *idx* et *pno*, respectivement. Ces données permettent d'adresser la table *Scb*. La comparaison de cette adresse *pno* avec les différentes bornes, basses et hautes, de chaque élément de liste, Ra_1 à Ra_Z, (adresses "*pno_start*" et "*pno_end*"), permet de déterminer s'il y a lieu d'appliquer une politique d'allocation de mémoire spécifique à la plage virtuelle concernée, et, si oui, quelle type de politique doit être appliquée ("*policy_type*"), et éventuellement quel numéro de module est concerné ("*module*"), selon le type précisé par le champ "*policy_type*".

Ces données supplémentaires sont initialisées lors de la création d'un segment, en fonction du profil de l'application concernée. Elles peuvent ensuite être modifiées par l'application de façon dynamique, au travers de l'instruction précitée de type "*system policy*".

Pour fixer les idées et à titre d'exemple, on a fait les suppositions suivantes :

- la politique globale à appliquer au segment est du type "P_LOCAL", et donc un numéro de module est inutile, puisque l'allocation s'effectue dans le module où a eu lieu la faute de page ;

- la politique spécifique associée à la première plage virtuelle, c'est-à-dire celle enregistrée dans le premier élément de la liste LRa_1, comprend les champs suivants : "*policy_type*" = "P_FIXE" et "*module*" = 2 ;

- la politique spécifique associée à la deuxième plage virtuelle, c'est-à-dire celle enregistrée dans le deuxième élément de la liste LRa_2, comprend les champs suivants : "*policy_type*" = "P_STRIPE" et "*module*" = 3 ;

....

- la politique spécifique associée à la plage virtuelle z, c'est-à-dire celle enregistrée dans le dernier élément de la liste LRa_z, comprend les champs suivants : "*policy_type*" = "P_DEFAULT" et "*module*" = "" (c'est-à-dire vide).

Naturellement les champs d'adresses "*pno_start*" et "*pno_end*" sont également documentés pour chacune des plages virtuelles, 1 à z.

Si l'adresse *pno* pointe un espace déterminé par les champs d'adresse de l'élément de liste LRa_2, le gestionnaire H reçoit, en réponse à sa requête, des données indiquant un type "P_STRIPE" et un numéro de module 3, soit M_C dans l'exemple de la figure 7a. Ce numéro de module doit être incrémenté (ou de façon plus générale modifié), puisqu'il s'agit d'une allocation tournante par bandes.

Le procédé conforme à la variante qui vient d'être décrite peut encore être amélioré. Il est en effet possible d'augmenter les performances du système, en diminuant le temps nécessaire pour déterminer la politique d'allocation de mémoire à appliquer pour une plage virtuelle, sur détection d'une faute de page. Pour ce faire, dans une variante du mode de réalisation supplémentaire qui vient d'être décrit, on utilise une table de plages virtuelles, à adressage calculé, c'est-à-dire à code dit de "hash" selon la terminologie anglo-saxonne.

La détermination d'une politique spécifique d'allocation s'effectue selon le diagramme des figures 10a à 10c. Les segments, par exemple le segment Sg_y, y étant un indice arbitraire, sont subdivisés en N sous-segments de longueurs fixes. Dans l'environnement "UNIX" précité, un segment représentant 256 MO de mémoire virtuelle, on le subdivise avantageusement en 256 sous-segments de 1 Mo chacun, référencés SS_1 à SS_N sur la figure 10b, ce qui est avantageux, puisqu'il s'agit d'une puissance de 2.

La table de "hash" *Th* (figure 10a) comprend également *N* entrées correspondant à *N* sous-segments de mémoire. Chacune des entrées stocke la politique d'allocation de mémoire spécifique aux sous-segments, SS_1 à SS_N. À chacun des emplacements mémoires de la table *Th* est associée une structure de liste individuelle. Plus exactement, cette structure de liste existe si, et seulement si, le sous-segment concerné comprend au moins une zone appartenant à une plage de mémoire virtuelle, ou "range", à laquelle une politique d'allocation de mémoire spécifique doit être appliquée.

Si on se reporte à la figure 10b, on a supposé que "*prange*" pour le segment Sg_Y était différent de zéro, donc que ce segment comprenait au moins une plage virtuelle associée à une politique d'allocation mémoire spécifique.

On a supposé que le sous-segment SS_1 ne comportait aucune plage virtuelle associée à une politique spécifique. C'est également le cas, sur la figure 10b, des sous-segments SS_3 et SS_N. Par contre, on a supposé que le sous-segment SS_2 comprenait trois plages, ou "ranges", Ra_{02} à Ra_{22}. La première, Ra_{02}, n'est pas associée à une politique spécifique, les deux autres, Ra_{12} à Ra_{22}, sont associées à des politiques d'allocations de mémoire spécifiques, par exemple "P_FIXE" et "P_STRIPE", respectivement.

Il s'ensuit que l'entrée e_2 de la table de "hash" *Th* est associée à une structure de liste comprenant deux éléments, LRa_{12} et LRa_{22}, emmagasinant les caractéristiques des deux politiques spécifiques, de la manière qui a été précédemment décrite.

Sur la figure 10a, il a été supposé que les entrées e_1, e_3, e_5 et e_6 et e_N, correspondent à des sous-segments de (même rang) qui ne comprennent pas de plages virtuelles associées à des politiques spécifiques. Par contre, outre l'entrée e_2 déjà citées, les entrées e_4 et e_7 correspondent à des sous-segments comprenant des plages virtuelles

associées à des politiques d'allocation de mémoire spécifiques. On constate que le nombre d'éléments de chaque liste individuelle, c'est-à-dire associée à une entrée, est variable. En effet, comme il a été indiqué, les plages virtuelles, ou "ranges", n'ont pas une longueur fixe. Dans l'exemple illustré sur la figure 10a, la structure de liste associée à l'entrée e_2 comporte deux éléments, LRa_{12} et LRa_{22}, la structure de liste associée à l'entrée e_4 comporte un élément, LRa_{14}, et la structure de liste associée à l'entrée e_7 comporte trois éléments, LRa_{17} à LRa_{37}.

Lorsqu'une faute de page est détectée, le gestionnaire *H* connaît l'adresse virtuelle qui a provoqué cette faute de page dans un segment donné, par exemple le segment d'indice *idx* et l'adresse *pno* qui permet de trouver le rang du sous-segment, par exemple le sous-segment SS_2.

Lors d'une première étape, l'entrée correspondante de la table *Th* est lue. Il existe une forte probabilité que la politique à appliquer puisse être déterminée directement à cette étape, par la lecture du jeu d'informations enregistrés dans la table *Th*. Si tel n'est pas le cas, seuls les éléments de la liste associés à une entrée déterminée, correspondant au sous-segment ayant causé la faute de page sont lus, c'est-à-dire les éléments associés à l'entrée N° 2 en l'occurrence. Ces éléments sont, à chaque fois en nombre restreint, car ils ne couvrent qu'un sous-segment, c'est-à-dire seulement 1 MO dans l'exemple décrit. Comme il a été indiqué, dans l'application préférée, sous environnement "UNIX", la granularité minimale d'une plage virtuelle, ou "range", est celle de la page. Le nombre maximum de plages par sous-segment est donc, au plus, limité au nombre de pages comprises dans un sous-segment (256 dans l'exemple).

Le processus d'acquisition des données nécessaire à la détermination d'une politique d'allocation de mémoire à appliquer est donc accéléré du fait des deux caractéristiques qui viennent d'être rappelés.

Les plages virtuelles, comme il a été rappelé, ne sont pas de longueurs fixes. Certaines plages pourraient alors se trouver "à cheval" sur deux sous-segments consécutifs, voire plus. C'est le cas d'une plage s'étendant sur 50 MO par exemple, si un sous-segment a une longueur de 1 MO. Ce problème peut être résolu en considérant que les plages peuvent elles-mêmes être subdivisées en sous-plages, ou "sub-ranges". Cette méthode n'est pas pénalisante car, au moment de leur création, le temps d'éxécution n'est pas critique. lors de la détection d'une faute de page, il est par contre nécessaire que l'attribution d'un emplacement de mémoire physique soit très rapide.

Dans le cadre de la variante du mode de réalisation supplémentaire qui vient d'être décrite, et si on se réfère maintenant à la figure 10c, on suppose, à titre d'exemple, qu'une faute de page a eu lieu pour une adresse virtuelle comprise dans un sous-segment donné. On suppose que la plage virtuelle lors de sa création a été subdivisée et répartie sur deux sous-segments consécutifs, de rangs arbitraires p et $p+1$.

Dans ce cas, les entrées p et $p+1$ de la table de "hash" *Th* sont toutes deux initialisées, au travers de l'instruction précitée de type "*system policy*", lors de la création de la plage d'adresses virtuelles. Lors d'une faute de page, une seule entrée de la table de "*hash*" *Th* est utilisée. Il s'agit de celle associée à cette faute de page, c'est-à-dire encore celle correspondant à un sous-segment (adresse *pno*).

S'ils existent, les éléments de liste associés à ces entrées, $LRa1p$, etc., et $LRa1(p+1)$, etc., respectivement, sont scrutés pour déterminer la politique d'allocation de mémoire à appliquer pour l'adresse ayant causé la faute de page précitée.

De façon plus générale, on utilise des entrées multiples correspondant à plusieurs sous-segments et une

entrée unique lorsqu'une plage virtuelle donnée est entièrement comprise dans un sous-segment, c'est-à-dire dans un sous-espace virtuel de 1 MO, dans l'exemple décrit.

A la lecture de ce qui précède, on constate aisément que l'invention atteint bien les buts qu'elle s'est fixés.

Elle permet notamment d'adapter au mieux l'utilisation de l'espace mémoire aux besoins réels des applications, c'est-à-dire en tenant compte de leurs profils spécifiques et des différents types de mémoire qu'elles utilisent. En outre, dans le mode de réalisation supplémentaire, un degré d'optimisation plus important peut être obtenu en descendant à un niveau plus fin, c'est-à-dire au niveau de ce qui a été appelé "plage virtuelle" ou "range", au prix de modifications légères des applications utilisant cette possibilité. Toutefois, même si ce mode de réalisation préféré est effectivement implanté dans la machine, il n'est pas nécessaire de modifier tous les types d'applications. Les applications non modifiées peuvent s'exécuter tel quel et bénéficient de l'amélioration des performances autorisées par les premiers modes de réalisation du procédé, et la politique d'allocation de mémoire est la politique applicable au niveau des segments. On peut généralement se contenter de ne modifier que les applications pour lesquelles le gain de performances escompté est important : applications manipulant des données sur un espace virtuel qu'il est important de localiser, comme les "drivers" de disques et de réseau, etc.

Il doit être clair cependant que l'invention n'est pas limitée aux seuls exemples de réalisations explicitement décrits. Notamment, le procédé ne saurait se limiter au seul jeu de règles prédéfinies d'allocation de mémoire explicité dans la description.

Il doit être clair aussi que, bien que particulièrement adaptée pour des architectures multiprocesseurs de type "NUMA" précité, on ne saurait

cantonner l'invention à ce seul type d'applications. Le procédé de l'invention s'applique avantageusement à tout système de traitement de l'information dont les accès à la mémoire physique ne s'effectuent pas de façon uniforme, que cette mémoire soit distribuée ou non entre plusieurs machines ou modules.

Enfin, bien que particulièrement adapté à un système d'exploitation sous environnement "UNIX" ou similaire, il est clair que l'on ne peut cantonner le procédé de l'invention à ce seul environnement.

REVENDICATIONS

1. Procédé d'allocation d'emplacements de mémoire physique par mise en correspondance avec au moins une plage d'adresses contiguës de mémoire dans un espace d'adressage virtuel associée à une application logicielle déterminée ($Appli_X$), l'application ($Appli_X$) étant en cours d'exécution dans un système de traitement de l'information (1') comprenant une unité de mémoire à accès non uniforme (*Mem*) et utilisant plusieurs types de mémoire virtuelle (tyM_1-tyM_6), ladite mise en correspondance s'effectuant par scrutation d'une table de correspondance d'adresses, caractérisé en ce qu'il comprend une étape consistant à lier ladite application logicielle déterminée ($Appli_X$) à des règles d'allocation de mémoire (*Rg*) choisies parmi un jeu de règles prédéfinies, et en ce que, lorsque ladite table de correspondance d'adresses ne comporte pas d'entrée pour une plage d'adresses contiguës de mémoire d'adresse virtuelle associée à ladite application logicielle déterminée (*Applix*), il comprend une étape de génération d'une exception (*Fp*) et une étape subséquente d'allocation d'un emplacement (Z_1-Z_n) de mémoire physique selon une desdites règles d'allocation de mémoire (*Rg*), le choix de la règle étant assujetti au profil (pa_X) desdits types de mémoire virtuelle utilisés (tyM_1-tyM_6) par l'application logicielle déterminée ($Appli_X$).

2. Procédé selon la revendication 1, caractérisé en ce que lesdites plages d'adresses contiguës de mémoire virtuelle se subdivisant en plusieurs catégories provoquant des types d'exception (*Fp*) distincts, il comprend une étape supplémentaire consistant à déterminer ledit type d'exception (*Fp*), et en ce que ladite règle d'allocation de mémoire est une fonction de la combinaison dudit profil et dudit type exception (*Fp*).

3. Procédé selon les revendications 1 ou 2, caractérisé en ce que ledit espace d'adressage virtuel est organisé en segments (Sg_X, Sg_Y) et en ce que lesdits segments (Sg_X, Sg_Y) sont associés auxdites règles (Rg) d'allocation de mémoire.

4. Procédé selon l'une quelconque des revendications 1 à 3, caractérisé en ce que ledit système de traitement de l'information (1', 1") étant constitué d'au moins deux modules distincts (M_a-M_C), comprenant chacun au moins un processeur (10_a-13_a, 10_b-13_b) et une unité de mémoire physique dite locale (14_a, 14_b), les unités de mémoire situées en dehors d'un module étant dites éloignées, ledit jeu de règles prédéfinies (Rg) comprend au moins les règles d'allocation de mémoire spécifiques suivantes, sur la génération d'une exception (Fp) :

- une première règle allouant un emplacement de mémoire exclusivement dans ladite unité de mémoire locale ;
- une deuxième règle allouant un emplacement de mémoire par distribution, selon des tranches, dans ladite unité de mémoire locale et lesdites unités éloignées ;
- et une troisième règle allouant un emplacement de mémoire fixe préétabli, dans ladite unité de mémoire locale ou lesdites unités éloignées, par référence à un numéro affecté auxdits modules.

5. Procédé selon la revendication 4, caractérisé en ce qu'il comprend au moins une règle supplémentaire d'allocation de mémoire prédéterminée, dite par défaut, de manière à ce que, lorsque ladite application logicielle déterminée ($Appli_X$) ayant provoqué une exception (Fp) n'est liée à aucune desdites règles spécifiques,

l'allocation de mémoire s'effectue selon ladite règle par défaut.

6. Procédé selon la revendication 4, caractérisé en ce que ladite application logicielle déterminée (*Applix*) comportant au moins un segment de mémoire partagé avec d'autres applications, accédé de façon sensiblement égale par l'ensemble desdits modules (M_a, M_b), ladite étape subséquente d'allocation d'un emplacement de mémoire physique (*Mem*) s'effectue conformément à ladite deuxième règle, l'allocation étant effectuée par distribution sur lesdites unités de mémoire locale et éloignées (14a, 14b).

7. Procédé selon la revendication 4, caractérisé en ce que ladite application logicielle déterminée (*Applix*) comportant au moins un segment de mémoire de travail, accédé en local dans l'un desdits modules (M_a-M_c), ladite étape subséquente d'allocation d'un emplacement de mémoire physique (*Mem*) s'effectue conformément à ladite première règle, l'allocation étant effectuée exclusivement dans ladite unité de mémoire locale (14a ou 14b).

8. Procédé selon l'une quelconque des revendications 4 à 7, caractérisé en ce que lesdits segments étant subdivisés en plages d'adressage virtuelles (Ra_1-Ra_5), chacune étant allouée à au moins l'une desdites applications (*Applix*), en ce qu'il est prévu une première donnée numérique spécifiant s'il existe au moins une plage d'adressage virtuel (Ra_1-Ra_5) à laquelle est associée à une règle d'allocation de mémoire spécifique, et en ce que ladite politique comprend au moins une deuxième donnée numérique spécifiant la nature de ladite règle d'allocation de mémoire et une troisième donnée numérique consistant en un numéro optionnel adressant l'un desdits modules (M_a-M_c).

9. Procédé selon la revendication 8, caractérisé en ce qu'il comprend, lors de ladite génération d'une exception (*Fp*) concernant une adresse comprise à l'intérieur de ladite plage d'adressage virtuelle (*Ra*$_1$-*Ra*$_5$), une étape de lecture de ladite première donnée numérique, et une étape subséquente consistant à allouer un emplacement de mémoire physique conformément aux règles d'allocation de mémoire régissant le segment (*Sg*$_X$, *Sg*$_Y$) lorsque ladite exception se traduit par un adressage de la mémoire ne correspondant à aucune desdites plages d'adressage virtuelle (*Ra*$_1$-*Ra*$_5$).

10. Procédé selon la revendication 8, caractérisé en ce que ladite deuxième donnée numérique spécifie au moins l'une des règles suivantes :

- une première règle allouant un emplacement de mémoire exclusivement dans ladite unité de mémoire locale ;
- une deuxième règle allouant un emplacement de mémoire par distribution, selon des tranches, dans ladite unité de mémoire locale et lesdites unités éloignées ;
- une troisième règle allouant un emplacement de mémoire fixe préétabli, dans ladite unité de mémoire locale ou lesdites unités éloignées ;
- et une quatrième règle, dite par défaut, allouant un emplacement de mémoire selon une politique d'allocation de mémoire globale audit système de traitement de l'information, ladite quatrième règle étant du type de l'une desdites première à troisième règles.

11. Procédé selon la revendication 10, caractérisé en ce que, lorsque ladite deuxième donnée numérique spécifie lesdites deuxième ou troisième règles d'allocation de mémoire, ladite troisième donnée consiste

en un numéro de module (M_a-M_c) à partir duquel est déterminé le module (M_a-M_c) dans lequel doit être effectué ladite allocation de mémoire.

12. Procédé selon les revendications 10 ou 11, caractérisé en ce que, ledit système de traitement de l'information comprenant une table de description des segments (*Scb*) comportant des entrées en nombre égal auxdits segments (Sg_X, Sg_Y) dudit espace virtuel, il comprend une étape initiale d'enregistrement desdites politiques d'allocation de mémoire dans ladite table de description de segments (*Scb*) et une étape initiale d'enregistrement desdits profils (Pa_X) dans des espaces de mémoire virtuelle occupés par lesdites applications ($Appli_X$) qui leur sont associées.

13. Procédé selon la revendication 12, caractérisé en ce que chacun desdits enregistrements de politique d'allocation de mémoire (Mp_Y) comprend au moins un premier champ stockant ladite première donnée numérique, un deuxième champ stockant ladite deuxième donnée numérique et un troisième champ stockant ladite troisième donnée numérique.

14. Procédé selon la revendication 13, caractérisé en ce que lesdits enregistrements comprennent deux champs supplémentaires, un premier champ stockant une donnée numérique spécifiant la borne d'adresse inférieure dans un segment déterminé (Sg_X, Sg_Y) de ladite plage d'adressage virtuel et un second champ stockant une donnée numérique spécifiant la borne d'adresse supérieure de cette plage d'adressage virtuel, et en ce qu'il comprend les phases suivantes :

- une phase préliminaire consistant à créer, à la demande desdites applications, pour chaque segment comprenant au

moins une plage virtuelle associée à une politique spécifique d'allocation de mémoire, une structure de liste comprenant des éléments en cascade (LRa_1-LRa_Z) en nombre égal auxdites plages d'adressage virtuel (Ra_1-Ra_5) comprises dans ledit segment (Sg_X, Sg_Y), chaque élément stockant lesdits enregistrements de politique d'allocation de mémoire ainsi que lesdits premier et second champ supplémentaire d'adresse, et étant associé à l'une des plages d'adressage virtuel (Ra_1-Ra_5) ;

- une phase subséquente, lors de la génération d'une exception (FP) à une adresse comprise dans un segment déterminé (Sg_X, Sg_Y) comprenant au moins les étapes suivantes :

a/ des étapes successives consistant en la lecture des données numériques stockées dans lesdits éléments de la structure de liste en cascade (LRa_1-LRa_Z) ;

b/ pour chacune des éléments de liste, une étape consistant en la comparaison de ladite adresse ayant provoqué l'exception (FP) avec lesdites bornes d'adresses inférieure et supérieure ;

c/ en cas de comparaison positive, une étape de lecture desdites deuxième et troisième données numériques, de manière à générer une instruction d'allocation de mémoire physique, conforme à ladite règle, et effectuée dans le module (M_a-M_C) dont le numéro est spécifié optionnellement en fonction de cette règle, ou en l'absence de règle spécifiée par ladite deuxième donnée numérique, à allouer un emplacement de mémoire physique conformément aux règles d'allocation de mémoire régissant le segment (Sg_X, Sg_Y) incluant la plage d'adressage virtuelle (Ra_1-Ra_5).

15. Procédé selon la revendication 13, caractérisé en ce que lesdits enregistrements comprennent deux champs

supplémentaires, un premier champ stockant une donnée numérique spécifiant la borne d'adresse inférieure dans un segment déterminé de ladite plage d'adressage virtuel (Ra_1-Ra_5) et un second champ stockant une donnée numérique spécifiant la borne d'adresse supérieure de cette plage d'adressage virtuel (Ra_1-Ra_5), et en ce qu'il comprend les phases suivantes :

- une première phase préliminaire supplémentaire comprenant les étapes suivantes :

1/ une première étape consistant à subdiviser chacun desdits segments (Sg_Y) comprenant au moins une plage virtuelle associée à une politique spécifique d'allocation de mémoire en sous-segments (SS_1-SS_N) de mêmes longueurs fixes ; et

2/ une deuxième étape consistant à créer une table (Th) comprenant autant d'entrées (e_1-e_N) que de sous-segments (SS_1-SS_N) ;

- une deuxième phase préliminaire supplémentaire consistant à créer, pour chaque sous-segment (SS_1-SS_N), associé à chacune desdites entrées, une structure de liste (LRa_{12}-LRa_{22}, LRa_{14}, LRa_{17}-LRa_{37}) comprenant des éléments en cascade en nombre égal auxdites plages d'adressage virtuel (Ra_{02}-Ra_{22}) comprises dans le sous-segment (SS_1-SS_N), chaque élément stockant lesdits enregistrements de politiques d'allocation de mémoire ainsi que lesdits premier et second champ supplémentaires d'adresse, et étant associé à l'une des plages d'adressage virtuel (Ra_{02}-Ra_{22}) ;

- une phase subséquente, lors de la génération d'une exception à une adresse comprise dans un sous-segment (SS_1-SS_N) déterminé comprenant au moins les étapes suivantes :

- a/ la lecture de ladite entrée de la table (Th) associée à liée à l'adresse ayant causé ladite exception et la détermination à partir de cette lecture si ledit sous-segment (SS_1-SS_N) inclue ou non une plage d'adressage virtuel (Ra_1-Ra_5) régie par une politique d'allocation de mémoire spécifique, et en cas de détermination négative, l'utilisation de la règle régissant ledit segment (Sg_Y) ;

- b/ en cas de détermination positive, des étapes successives consistant la lecture des données numériques stockées dans lesdits éléments de la structure de liste en cascade (LRa_{12}-LRa_{22}, LRa_{14}, LRa_{17}-LRa_{37}) associée à l'entrée liée à l'adresse ayant causé ladite exception ;

- c/ pour chacune des éléments de liste (LRa_{12}-LRa_{22}, LRa_{14}, LRa_{17}-LRa_{37}), une étape consistant en la comparaison de ladite adresse ayant provoqué l'exception avec lesdites bornes d'adresses inférieure et supérieure ;

- d/ en cas de comparaison positive, une étape de lecture desdites deuxième et troisième données numériques, de manière à générer une instruction d'allocation de mémoire physique conforme à ladite règle et effectuée dans le numéro de module (M_a-M_C), spécifié optionnellement en fonction de cette règle.

16. Procédé selon la revendication 15, caractérisé en ce que lesdites plages d'adressage de mémoire virtuelle (Ra_{02}-Ra_{22}) étant de longueurs variables, lorsque ladite longueur est supérieure à la longueur desdits sous-segments (SS_1-SS_N) de longueur fixe, lesdites plages d'adressage de mémoire virtuelle sont subdivisées en sous-espaces compris dans des sous-segments (SS_1-SS_N).

17. Procédé selon les revendications 15 ou 16, caractérisé en ce que lesdits segments (Sg_Y) s'étendent sur un espace virtuel contigu de 256 MO et en ce que ladite table (Th) comporte 256 entrées (e_1-e_N), chacune correspondant à un sous-segment (SS_1-SS_N) de 1 MO.

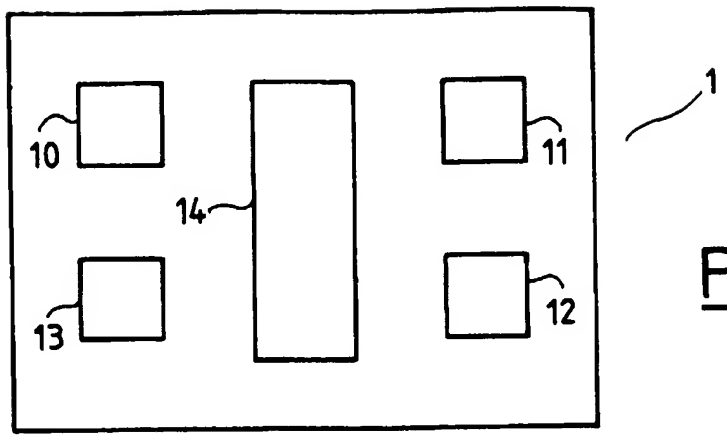


FIG_1

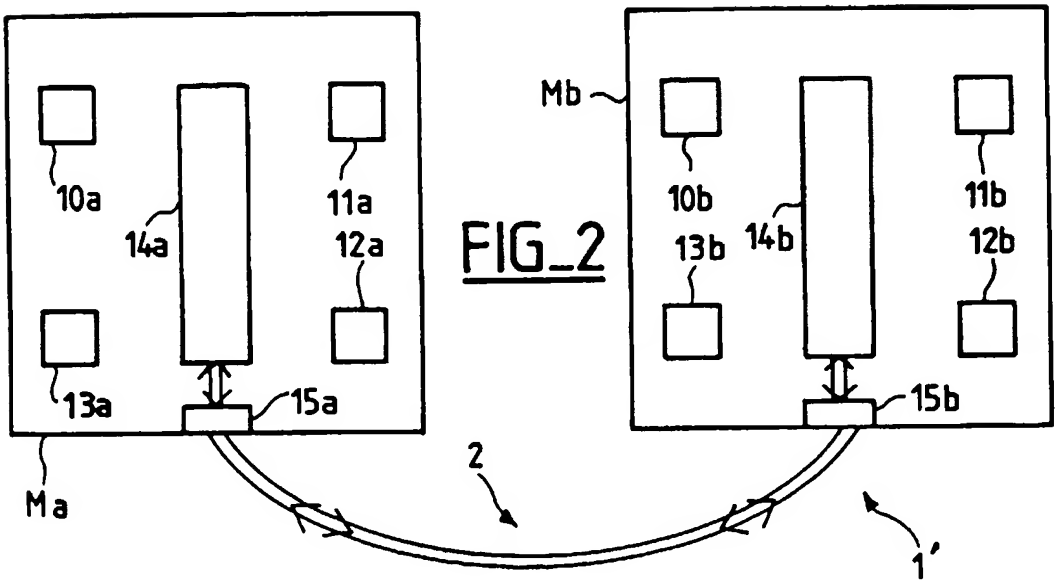


FIG_2

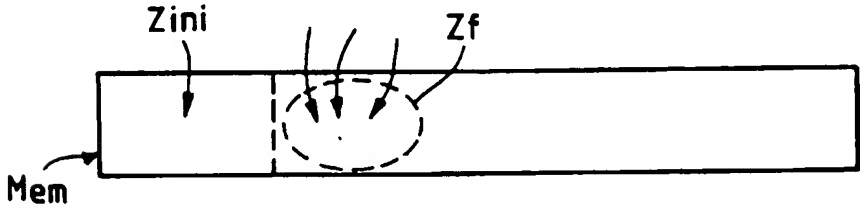


FIG_3a

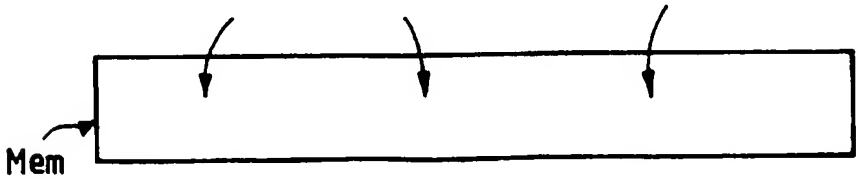


FIG_3b

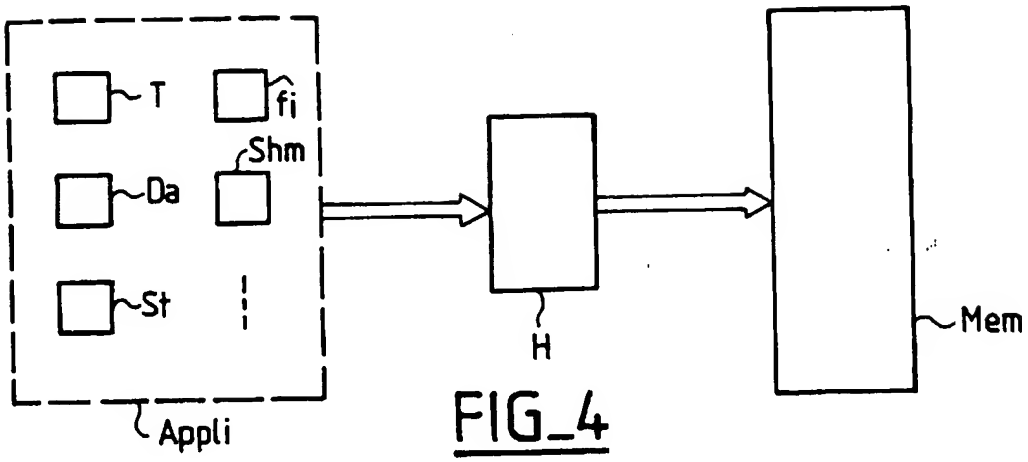


FIG_4

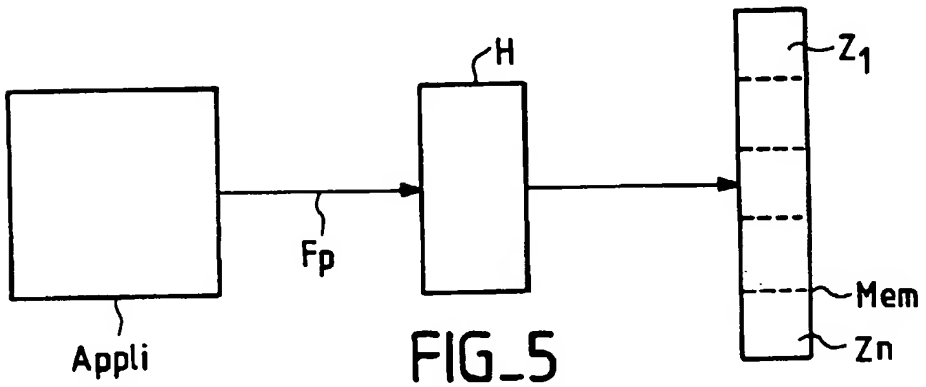


FIG_5

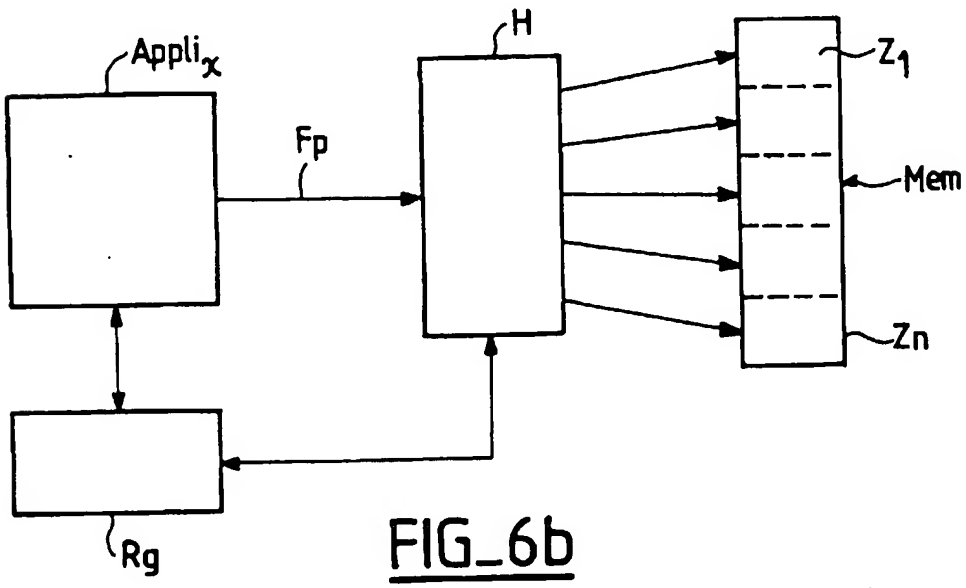


FIG_6b

FIG_6a

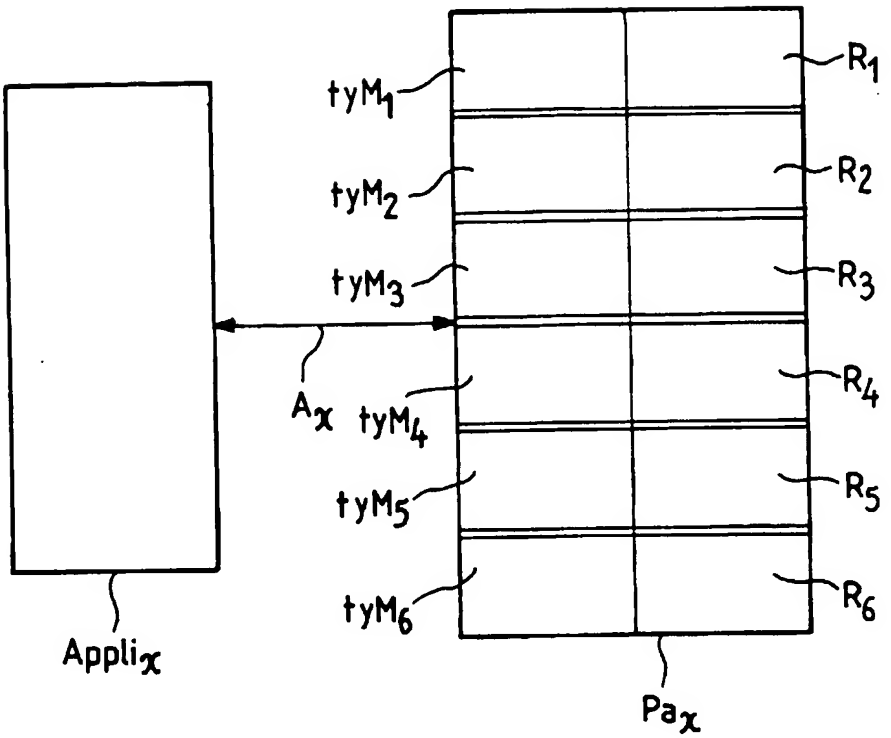
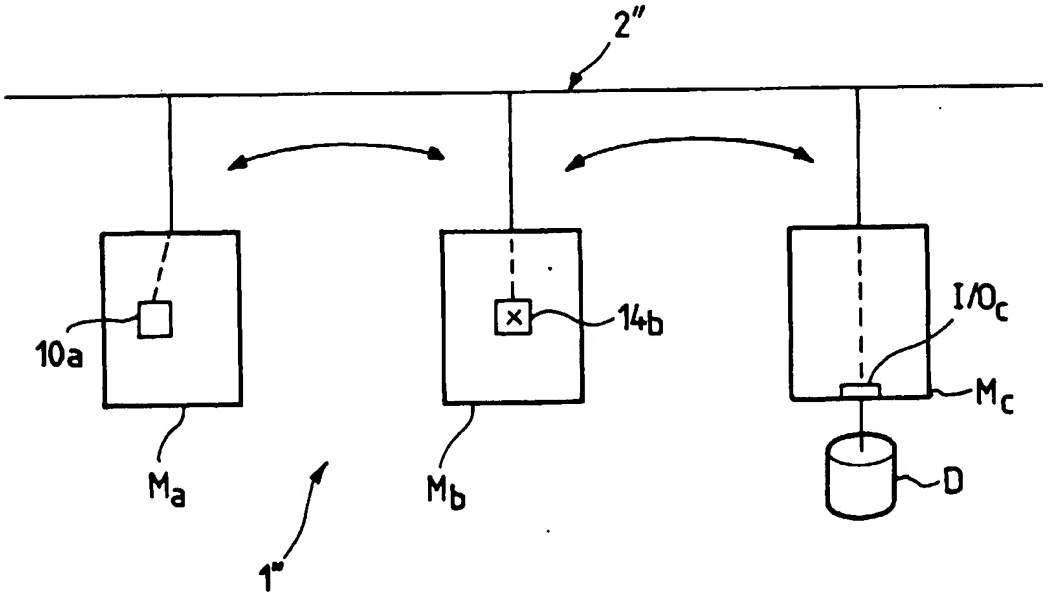

tyM1
tyM2
tyM3
tyM4
tyM5
tyM6
R1
R2
R3
R4
R5
R6
Ax
Applix
Pax

FIG_7a


2"
10a
14b
I/Oc
Mc
D
Ma
Mb
1"

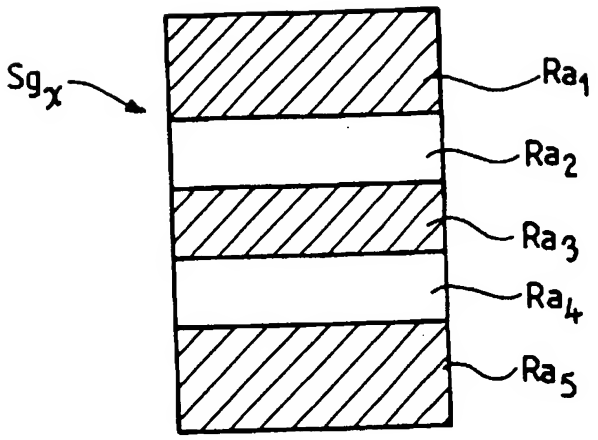


FIG_7b

FIG_8

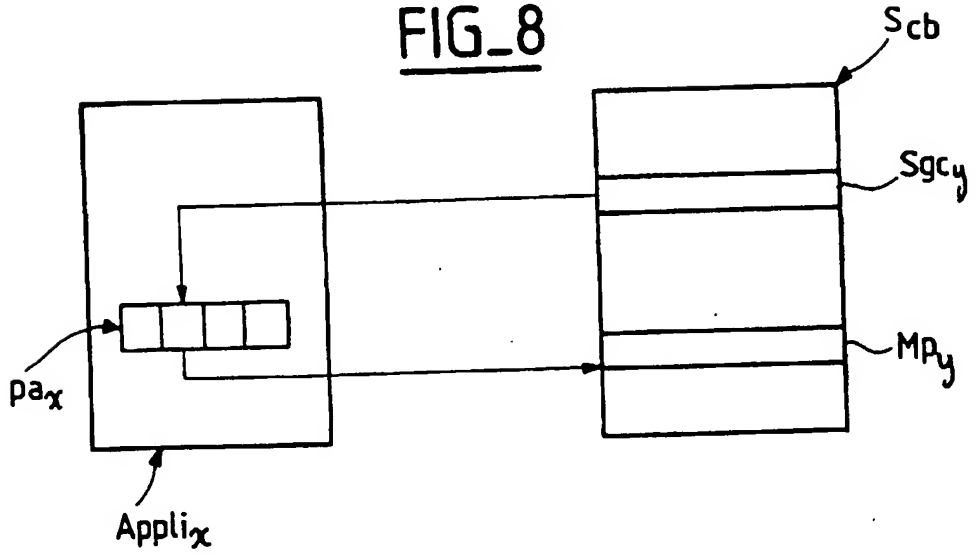
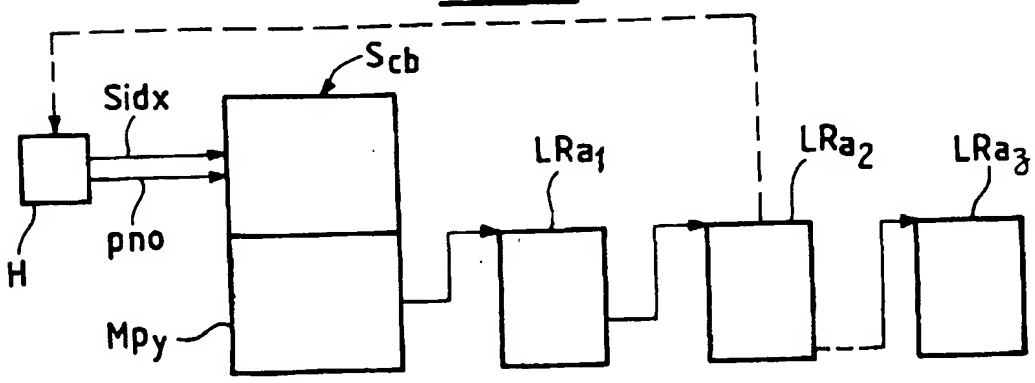


FIG_9



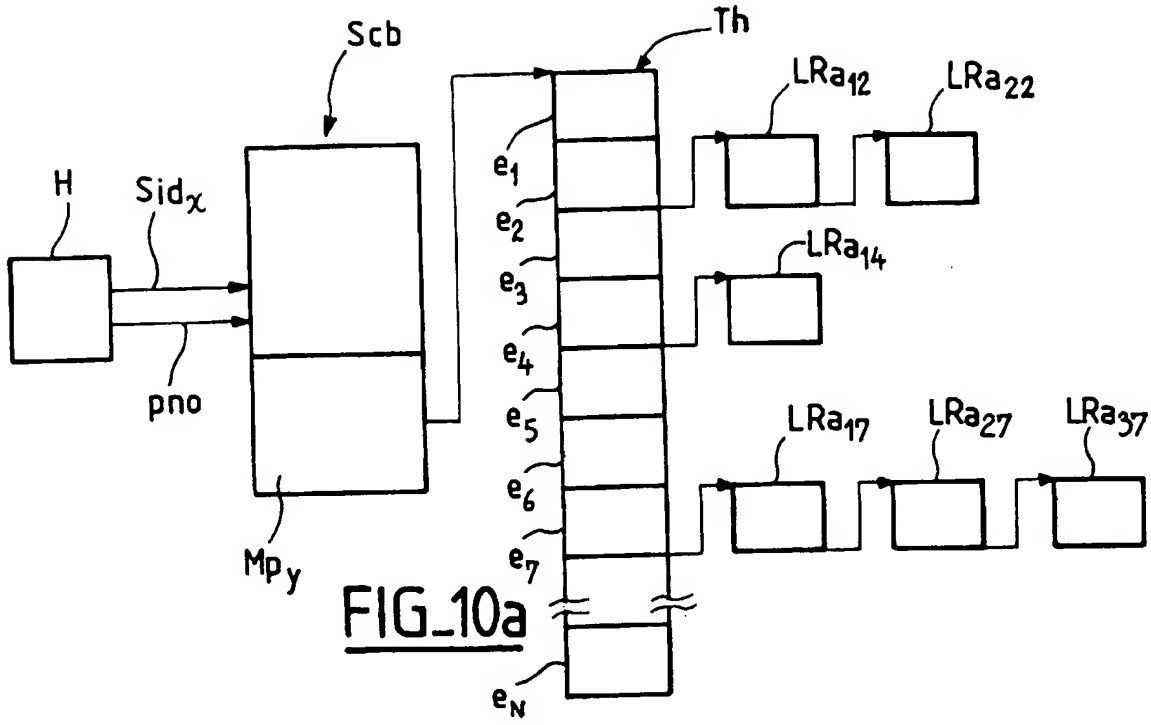


FIG_10a

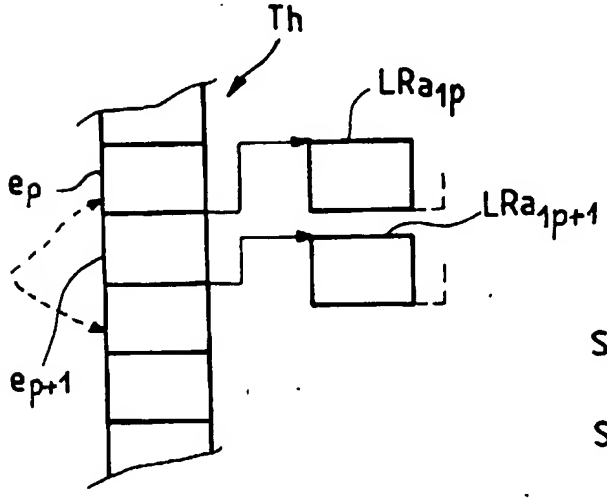
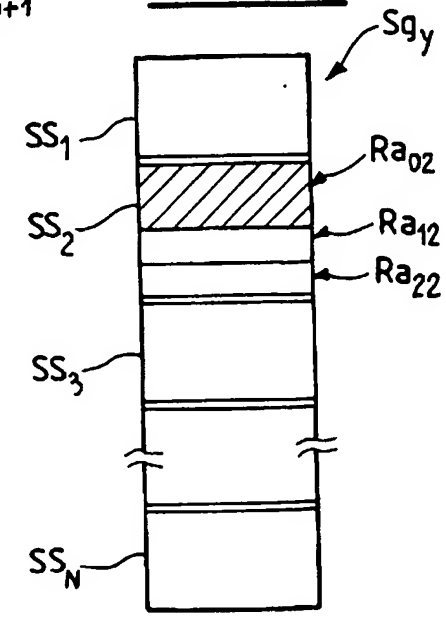


FIG_10c

FIG_10b



N°d'enregistrement national: **98 08058** N°de publication: 2767939

RAPPORT DE RECHERCHE

articles L.612-14, L.612-17 et R.612-53 à 69 du code de la propriété intellectuelle

OBJET DU RAPPORT DE RECHERCHE

Après l'accomplissement de la procédure prévue par les textes rappelés ci-dessus, le brevet est délivré. L'Institut National de la Propriété Industrielle n'est pas habilité, sauf dans le cas d'absence **manifeste** de nouveauté, à en refuser la délivrance. La validité d'un brevet relève exclusivement de l'appréciation des tribunaux.

L'I.N.P.I. doit toutefois annexer à chaque brevet un "RAPPORT DE RECHERCHE" citant les éléments de l'état de la technique qui peuvent être pris en considération pour apprécier la brevetabilité de l'invention. Ce rapport porte sur les revendications figurant au brevet qui définissent l'objet de l'invention et délimitent l'étendue de la protection.

Après délivrance, l'I.N.P.I. peut, à la requête de toute personne intéressée, formuler un "AVIS DOCUMENTAIRE" sur la base des documents cités dans ce rapport de recherche et de tout autre document que le requérant souhaite voir prendre en considération.

CONDITIONS D'ÉTABLISSEMENT DU PRÉSENT RAPPORT DE RECHERCHE

- ☒ Le demandeur a présenté des observations en réponse au rapport de recherche préliminaire.
- ☒ Le demandeur a maintenu les revendications.
- ☐ Le demandeur a modifié les revendications.
- ☐ Le demandeur a modifié la description pour en éliminer les éléments qui n' étaient plus en concordance avec les nouvelles revendications.
- ☐ Les tiers ont présenté des observations après publication du rapport de recherche préliminaire.
- ☐ Un rapport de recherche préliminaire complémentaire a été établi.

DOCUMENTS CITÉS DANS LE PRÉSENT RAPPORT DE RECHERCHE

La répartition des documents entre les rubriques 1, 2 et 3 tient compte, le cas échéant, des revendications déposées en dernier lieu et/ou des observations présentées.

- ☒ Les documents énumérés à la rubrique 1 ci-après sont susceptibles d'être pris en considération pour apprécier la brevetabilité de l'invention.
- ☒ Les documents énumérés à la rubrique 2 ci-après illustrent l'arrière-plan technologique général.
- ☐ Les documents énumérés à la rubrique 3 ci-après ont été cités en cours de procédure, mais leur pertinence dépend de la validité des priorités revendiquées.
- ☐ Aucun document n'a été cité en cours de procédure.

BT 563/040496

N° d'enregistrement national : 98 08058 N° de publication : 2767939

1.ELEMENTS DE L'ETAT DE LA TECHNIQUE SUSCEPTIBLES D'ETRE PRIS EN CONSIDERATION POUR APPRECIER LA BREVETABILITE DE L'INVENTION

Référence des documents (avec indication, le cas échéant, des parties pertinentes)	**Revendications du brevet concernées**
" PIPELINE ARTICLE 19970206 PERFORMANCE TUNING FOR NONUNIFORM MEMORY ARCHITECTURE ", MIS A JOUR 4 JUIN 1997 XP002086738 Accessible par Internet : <URL : http://heron.cc.ukans.edu/pipeline-article s/pipeline-numa.html> 3 décembre 1998 * le document en entier *	1-17
KRUEGER K ET AL : " TOOLS FOR THE DEVELOPMENT OF APPLICATION-SPECIFIC VIRTUAL MEMORY MANAGEMENT " ACM SIGPLAN NOTICES, Vol. 28, no. 10, 1er octobre 1993, Pages 48-64, XP000411717 * page 51, colonne de droite, ligne 1 - page 56, colonne de droite, ligne 26 ; figure 1 *	1-7
EP 0 750 255 A (DATA GENERAL CORP) 27 décembre 1996 * colonne 22, ligne 15 - colonne 28, ligne 5 *	1-7

2.ELEMENTS DE L'ETAT DE LA TECHNIQUE ILLUSTRANT L'ARRIERE-PLAN TECHNOLOGIQUE GENERAL

LAROWE JR R P ET AL : " PAGE PLACEMENT POLICIES FOR NUMA MULTIPROCESSORS " JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING, Vol. 11, no. 2, 1er février 1991, Pages 112-129, XP000201935

JAYASHREE RAMANATHAN ET AL : " CRITICAL FACTORS IN NUMA MEMORY MANAGEMENT * " INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, ARLINGTON, TEXAS, MAY 20 - 24, 1991, No. CONF. 11, 20 mai 1991, Pages 500-507, XP000221890 INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS

3. ELEMENTS DE L'ETAT DE LA TECHNIQUE DONT LA PERTINENCE DEPEND DE LA VALIDITE DES PRIORITES

Référence des documents (avec indication, le cas échéant, des parties pertinentes)	**Revendications du brevet concernées**
NEANT	